Refine Search

Search Results -

Terms	Documents
L6 and L5	1

Database:
- US Pre-Grant Publication Full-Text Database
- **US Patents Full-Text Database**
- US OCR Full-Text Database
- EPO Abstracts Database
- JPO Abstracts Database
- Derwent World Patents Index
- IBM Technical Disclosure Bulletins

Search: L7

Refine Search

Recall Text Clear Interrupt

Search History

DATE: Thursday, April 29, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; THES=ASSIGNEE; PLUR=YES; OP=OR			
L7	L6 and l5	1	L7
L6	l2 and (metadata same meanings)	18	L6
L5	l2 and (metadata same interpretation)	19	L5
L4	l2 and (metadata same meanings same interpretation)	0	L4
L3	L2 and (contract$ same manag$)	13	L3
L2	metadata and meanings and interpretation	126	L2
L1	6640145.pn.	1	L1

END OF SEARCH HISTORY

First Hit Fwd Refs

Generate Collection Print

L8: Entry 1 of 8 File: USPT Apr 27, 2004

DOCUMENT-IDENTIFIER: US 6728752 B1
TITLE: System and method for information browsing using multi-modal features

Brief Summary Text (21):
Much research in information retrieval has focused on retrieving text documents based on their textual content or on retrieving image documents based on their visual features. Moreover, with the explosion of information on the web and corporate intranets, users are inundated with hits when searching for specific information. The task of sorting through the results to find what is really desired is often tedious and time-consuming. Recently, a number of search engines have added functionality that permits users to augment queries from traditional keyword entries through the use of metadata (e.g., Hotbot, Infoseek). The metadata may take on various forms, such as language, dates, location of the site, or whether other modalities such as images, video or audio are present.

Brief Summary Text (25):
One difficulty in the use of multiple features in search and browsing is the combination of the information from the different features. This is commonly handled in image retrieval tasks by having weights associated with each feature (usually image features such as color histogram, texture, and shape) that can be set by the user. With each revision of the weights, a new search must be performed. However, in employing a heterogeneous set of multi-modal features, it is often difficult to assign weights to the importance of different features. In systems that employ metadata, the metadata usually has finite, discrete values, and a Boolean system that includes or excludes particular values can be used. Extending the concept to multi-modal features that may not be discrete leads exacerbates the question of how to combine the features.

Brief Summary Text (38):
It is envisioned that analogous to a database with various metadata fields, the documents in the present collection are characterized by many different features, or (probably non-orthogonal) "dimensions," many of which are derived from the contents of the unstructured documents.

Brief Summary Text (39):
Multi-modal features may take on many forms, such as user information, text genre, or analysis of images. The features used in the present invention can be considered a form of metadata, derived from the data (text and images, for example) and its context, and assigned automatically or semi-automatically, rather than current image search systems, in which metadata is typically assigned manually. Table 1 lists several possible features (all of which will be described in greater detail below); it will be recognized that various other features and modalities are also usable in the invention, and that the features of Table 1 are exemplary only.

Detailed Description Text (39):
A document's text genre is embedded into R.sup.n.sup..sub.g , where n.sub.g is the number of known text genres. A document genre is a culturally defined document category that guides a document's interpretation. Genres are signaled by the greater document environment (such as the physical media, pictures, titles, etc.

that serve to distinguish at a glance, for example, the National Enquirer from the New York Times) rather than the document text. The same information presented in two different genres may lead to two different interpretations. For example, a document starting with the line "At dawn the street was peaceful . . . " would be interpreted differently by a reader of Time Magazine than by a reader of a novel. Each document type has an easily recognized and culturally defined genre structure which guides our understanding and interpretation of the information it contains. For example, news reports, newspaper editorials, calendars, press releases, and short stories are all examples of possible genres. A document's structure and genre can frequently be determined (at least in part) by an automated analysis of the document or text (step 510). Although text genre might not always be determinable, particularly with web pages (which frequently do not have a well-defined genre), it is generally possible to calculate a vector of probability scores (step 512) for a number of known possible genres; that vector can then be used to determine similarity (via a cosine similarity computation) in the manner discussed above with regard to text term vectors: ##EQU6##

Detailed Description Text (97):
When using a clustering scheme such as Scatter/Gather, it is necessary to display or otherwise represent the clusters to the user during a browsing session. A text cluster can be represented in a number of ways, the most common being the selection and display of a set of words that are in some way most representative of the cluster. When image clusters need to be represented, it is less meaningful to choose image features that are common to the cluster members and display them, since these will not, in general, have semantic meaning to the user. Previous clustering image browsers have represented image clusters by mapping the images into a lower (two) dimensional space and displaying the map. Instead, a preferred embodiment of the invention calls for a further clustering of the cluster, followed by representing the cluster by (a) the three images closest to the centroid of the cluster, and (b) three images representative of subregions of the cluster. The three subregion representatives are computed by removing the three most central images from (a) above, computing three subclusters, and using the image closest to the centroid of each subcluster (as measured via the appropriate distance metric). This representation provides a sense of the cluster centroid and the range of images in the cluster. The representative images could also have been placed on a 2-D display using multi-dimensional scaling, but for the examples in this disclosure, the representatives are displayed in a row of three "centroid" images or three "subcluster" images (see, e.g., FIG. 14). This permits very similar images, such as thumbnails and multiple copies of originals, to be more readily identified.

Detailed Description Text (98):
A collection of Web-like documents containing 2,310 images has been used as an exemplary corpus for the examples set forth below. Web documents contain many of the same types of "meta-information" that can be found in scanned images of documents and can be used to infer the content of a document or the components in a document. By working with web documents, the issues involved with identifying components and layout in an image are minimized, while permitting development of techniques for using metadata in the retrieval process.

Detailed Description Text (123):
The main technique used for CUA as described herein is multi-modal clustering of users; however, there remains the issue of trying to interpret those clusters. In the abstract, the objects of a cluster are characterized by similarities among the objects on features of text, usage, collection topology (inlinks and outlinks), and URL. To reveal these characteristic similarities among objects, a variety of user interface and visualization techniques are employed.

First Hit Fwd Refs
End of Result Set

☐ Generate Collection Print

L11: Entry 1 of 1 File: USPT Nov 21, 2000

US-PAT-NO: 6151602
DOCUMENT-IDENTIFIER: US 6151602 A

TITLE: Database system with methods providing a platform-independent self-describing data packet for transmitting information

DATE-ISSUED: November 21, 2000

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Hejlsberg; Anders	Seattle	WA		
Hansen; Kurt	Aptos	CA		

ASSIGNEE-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY	TYPE CODE
Inprise Corporation	Scotts Valley	CA			02

APPL-NO: 09/ 047924 [PALM]
DATE FILED: March 25, 1998

PARENT-CASE:
The present application claims the benefit of priority from commonly-owned application Ser. No. 60/064,920, filed Nov. 7, 1997, the disclosure of which is hereby incorporated by reference.

INT-CL: [07] G06 F 17/30

US-CL-ISSUED: 707/10; 707/101, 709/201
US-CL-CURRENT: 707/10; 707/101, 709/201

FIELD-OF-SEARCH: 707/10, 707/101, 709/201

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	5293379	March 1994	Carr	370/474
☐	5483522	January 1996	Derby et al.	370/400
☐	5701302	December 1997	Geiger	370/521

☐	5805808	September 1998	Hasani et al.	709/243
☐	5930786	July 1999	Carino, Jr. et al.	707/4

OTHER PUBLICATIONS

Feldman, Phil (Using Visual Basic 3, Que, p. 1003), 1993

ART-UNIT: 271

PRIMARY-EXAMINER: Black; Thomas G.

ASSISTANT-EXAMINER: Trinh; William

ATTY-AGENT-FIRM: Smart; John S.

ABSTRACT:

A three-tier data processing system of the present invention includes a client application, operating on a client machine (i.e., first tier), which obtains data from a back-end data source (e.g., database server) by submitting a request (e.g., SQL query) to a middle tier. The middle tier, in turn, comprises a provider and a resolver. Data is actually returned to the client by means of a "data packet" of the present invention, which is a platform-independent self-describing data format used to exchange data between different subsystems of the architecture. A data packet normally represents a result set, which is received by a client from a remote server, containing both data and metadata. Upon receiving the data packet from the provider, the client unpacks the data and then proceeds to process and manipulate the data as if it were local data (e.g., for insert, deletes, updates, and the like). Additional data packets are provided for special purpose use, including a "delta" data packet, used when applying client updates, and an "error" data packet, used to report results (including errors, after a failed update attempt) back to a client. The resolver, upon receiving a delta data packet, applies logic for effecting the user-specified modifications to the data set present at the back end. In the event that the resolver is unable to apply the user-specified modifications, it reports the failed operation back to the client via an error data packet.

21 Claims, 7 Drawing figures

L11: Entry 1 of 1 File: USPT Nov 21, 2000

DOCUMENT-IDENTIFIER: US 6151602 A
TITLE: Database system with methods providing a platform-independent self-describing data packet for transmitting information

Abstract Text (1):
A three-tier data processing system of the present invention includes a client application, operating on a client machine (i.e., first tier), which obtains data from a back-end data source (e.g., database server) by submitting a request (e.g., SQL query) to a middle tier. The middle tier, in turn, comprises a provider and a resolver. Data is actually returned to the client by means of a "data packet" of the present invention, which is a platform-independent self-describing data format used to exchange data between different subsystems of the architecture. A data packet normally represents a result set, which is received by a client from a remote server, containing both data and metadata. Upon receiving the data packet from the provider, the client unpacks the data and then proceeds to process and manipulate the data as if it were local data (e.g., for insert, deletes, updates, and the like). Additional data packets are provided for special purpose use, including a "delta" data packet, used when applying client updates, and an "error" data packet, used to report results (including errors, after a failed update attempt) back to a client. The resolver, upon receiving a delta data packet, applies logic for effecting the user-specified modifications to the data set present at the back end. In the event that the resolver is unable to apply the user-specified modifications, it reports the failed operation back to the client via an error data packet.

Brief Summary Text (6):
Often, networks are configured as "client/server" networks, such that each computer on the network is either a "client" or a "server." Servers are powerful computers or processes dedicated to managing shared resources, such as storage (i.e., disk drives), printers, modems, or the like. Servers are often dedicated, meaning that they perform no other tasks besides their server tasks. For instance, a database server is a computer system that manages database information, including processing database queries from various clients.

Brief Summary Text (17):
A data packet binary image or layout includes a header, a variable number of column descriptors, a variable number of optional parameters, and a variable number of rows containing the actual row data. The column descriptors and the optional parameters together comprise the metadata for the data packet. The core metadata is provided by the column descriptors. Each column descriptor fully characterizes a particular column, including specifying a name and data type for the column. The optional parameters specify additional metadata, including descriptors describing any indexes on the data set as well as descriptors describing constraints (e.g., referential integrity and primary key constraints) on the data set. As another example, the optional parameters can specify that a data set is read only, which is helpful, for instance, when a data set is the result of a join operation between two or more tables. Since the data packet is extensible, the client application can add its own user-defined optional parameters (which are passed along with the data

packet).

Brief Summary Text (19):
A data packet, which normally contains both data and metadata, represents a result set received by a client from a data source (e.g., a remote server). Upon receiving the data packet, the client unpacks the data and then proceeds to process and manipulate the data as if it were local data (e.g., for insert, deletes, updates, and the like). Additional data packets are provided for special purpose use, including a "delta" data packet, used when applying client updates, and an "error" data packet, used to report results back to a client, including errors after a failed update attempt. Both of these data packets have the same layout and structure as a normal data packet, but the data content is interpreted in a slightly different manner. The resolver, upon receiving a delta data packet, applies logic for effecting the user-specified modifications to the data set present at the back end. In the event that the resolver is unable to apply the user-specified modifications, it generates an error data packet for communicating the results of its operations back to the client, including information about why the error occurred.

Detailed Description Text (19):
The data packet is a platform-independent self-describing data format, used to exchange data between different subsystems of the architecture. It is also used as a file format, when storing data temporarily to disk. A data packet normally represents a result set received by a client from a remote server, containing both data and metadata. Additional data packets are provided, however, for special purpose use. These include, for instance, a "delta" data packet, used when applying client updates, and an "error" data packet, used to report results and errors (e.g., a failed update attempt) back to a client. Both of these data packets have the same layout and structure as a normal data packet, but the data content is interpreted in a slightly different manner. Normally it will be clear from the context which kind of data packet is used, but the data packet also contains an indicator for its kind.

Detailed Description Text (23):
(2) Self-contained and self-describing: The data package itself contains all the information necessary for unpacking and interpreting the data. The structure of the data in a data packet is described using metadata. The metadata information includes column descriptors and special attributes. The column descriptors, which are based on a basic set of data types, are employed to characterize each column (e.g., by name and data type). Special attributes of the data are described using optional parameters. Special semantics of the data, like update semantics, constraints, and ordering, are likewise described using optional parameters. Through use of the metadata, the client has full knowledge of the data set, including understanding indexes on the data set (useful for catching key violations which occur at the client) as well as understanding referential integrity constraints present on the data set.

Detailed Description Text (24):
(3) Extensible: Optional parameters allow for new data or object types (i.e., user-defined types), or attributes to be added seamlessly, as long as both sender and receiver understands their meaning.

Detailed Description Text (27):
A data packet representing ordinary data can be "partial," meaning the total data content is divided into multiple data packets. In this case, only the first data packet contains metadata describing the data. The subsequent data packets merely include an data packet identifier (i.e., type partial, stored in the header) and data rows. Partial data packets are used to reduce data-traffic when the user just wants to see the first N number of rows of the results set, and later fetch the next set of rows in a partial data packet (e.g., as the user scrolls to the end).

If the user scrolls down the entire set, he or she will eventually assemble the full result set. In this manner, the partial data packet provides "on demand" data delivery.

Detailed Description Text (30):
The column descriptors and the optional parameters together comprise the metadata 420 for the data packet. The core metadata is provided by the column descriptors. Each column descriptor fully characterizes a particular column, including specifying a name and data type for the column. The optional parameters specify additional metadata, including descriptors describing any indexes on the data set as well as descriptors describing entity integrity (i.e., primary key) and referential integrity constraints on the data set. As another example, the optional parameters can specify that a data set is read only, which is helpful, for instance, when a data set is the result of a join operation between two or more tables. Since the data packet is extensible, a component of the system (e.g., application server or client application) can register its own user-defined optional parameters (which are passed along with the data packet).

Detailed Description Text (42):
Here, the iMagicCookie field allows quick validation of a data packet by checking that the first four bytes of the image contain the expected value. By including a iHeaderSize field in the header, additional fields can be added to the fixed part of the header in future versions. The beginning of the column descriptor section (described below) is found by adding the iHeaderSize field to the beginning of the header. The iColCount and iRowCount fields give the number of columns and rows in the data packet. The data packet might not contain any metadata (column descriptors, and/or optional parameters) if it is a partial data packet (i.e., iProperties contains the pcklpropsMETADATA.sub.-- INCL bit). The first data packet in a series of partial data packets, should contain metadata information, but does not need to contain any data.

Detailed Description Text (48):
As previously mentioned, a data packet includes an area called the optional parameters section, following the column descriptor section, which contains additional metadata information, like constraints, data-ordering information, and the like. In addition, each column descriptor is followed by an (possibly empty) optional parameters section.

Detailed Description Text (65):
The error data packet is returned from a resolver to report the results, including informing the client of an update request failure. The error data packet contains one row for each failed update-request. The rows have a number of predefined columns, followed by all the columns of the original data packet. The latter are used to return the values of a conflicting row, if any, to the client. The following table outlines the meaning of the predefined columns used for error data packets.

Detailed Description Text (67):
In a "partial" data packet the total data content is divided into multiple data packets. In order to reduce overhead, only the first data packet contains metadata describing the data. The subsequent data packets merely contain the fixed size header and actual data contained in data-rows. The iProperties of the fixed size header, indicates whether a data packet contains metadata or not (pcklpropsMETADATA.sub.-- INCL).

Detailed Description Text (96):
Referring now to FIG. 5, a flow chart 500 summarizing the overall methodology of the present invention is illustrated. At step 501, a client generates a request (e.g., SQL statement) for data from a data source, such as a relational database table stored on a back end server. In response to this request, at step 502, the

provider accesses the data from the data source; it may now proceed to construct a data packet. In particular, the provider creates a data packet header comprising a metadata and optional parameters. Since stream I/O is employed, the provider can sequentially read column descriptor information from the data source (e.g., cursor handle on a result table) and then stream out corresponding metadata. After the column descriptor information has been added to the stream, at step 503, the provider proceeds to gather other information, such as relevant indexes and constraints (e.g., from the cursor). This information is added one by one to the stream as optional parameters, at step 504. In an exemplary embodiment, each attribute is added in the form of an attribute name following by an attribute value (e.g., as it in the manner used for Microsoft Windows .ini file). After all attribute information has been streamed out, the system may begin reading and processing actual data.

Detailed Description Text (98):
Processing at the client occurs in a similar manner, on a first-in, first-out basis, as indicated by flow chart 600 in FIG. 6. After the data packet is transmitted to the client, at step 601, the client starts the unpacking process by reading the data packet header for processing the column descriptors (step 602) and optional parameters (step 603). With this information, the client can set up a local data store for receiving the actual data, as indicated at step 604. Thereafter, the client can proceed to process the individual data records, at step 605, using the metadata information for correctly interpreting the stream. In this manner, the data is correctly reconstituted at the client (e.g., in a local data store). Processing of the stream itself terminates upon encountering the "end of stream" token, at step 606. Now, the data exists at the client and the data packet may be discarded. From the point of view of the client application, the data appears the same as if it were a local table. The client at this point would typically return a handle or cursor to a local table, for receiving updates from that client, as shown in step 607.

Detailed Description Text (99):
In addition to the regular data packet outlined above, the system employs a "partial" data packet in those instances where it is necessary to spread the data out among multiple data packets. Consider, for instance, a scenario where the client only desires the first 50 data records of a result set. Here, a regular data packet can be sent to the client containing those first 50 records. Should the client then request the next 50 records (e.g., "fetch next"), a partial data packet can be sent containing only those next data records; in particular, the metadata information is not transmitted since the context is already known at that time.

Detailed Description Paragraph Table (2):
TABLE 2 ______________________________________
Type descriptor format Bits Meaning Description
______________________________________ 0 . . .
15 Size Size indicator (1 . . . 64k). For a fixed size type this field gives the size in bytes of the data. For a varying size type this field gives the size in bytes (1, 2, or 4) of the byte count (8-, 16-, or 32-bit) that prefixes the data. 16 . . . 21 Type Type indicator (0 . . . 63). This field indicates how the data should be interpreted. 22 Varying 0 = Fixed size type, 1 = Varying size type. For a fixed size type, the Size field gives the size in bytes of the data. For a varying size type, the Size field gives the size in bytes (1, 2, or 4) of the length indicator that prefixes the data. 23 Array 0 = Single element, 1 = Array. For an array, the data consists of a 32-bit element count followed by that many elements. 24 . . . 31 Unused
____

Detailed Description Paragraph Table (4):
TABLE 4 ______________________________________ Type code examples Type code Data format ______________________________________ 0.times.00010001 1 byte to be

interpreted as an 8-bit signed integer. 0.times.00020004 4 bytes to be interpreted as a 32-bit unsigned integer. 0.times.00490001 1 byte length indicator followed by that many bytes of data to be interpreted as a multi-byte string value. 0.times.004A0002 2 byte length indicator followed by that many bytes of data to be interpreted as a Unicode string (with half as many Unicode characters). 0.times.004B0004 4 byte length indicator followed by that many bytes of data to be interpreted as a block of bytes. 0.times.00820002 4 byte element count followed by that many 1 6-bit unsigned integers. ______________________________________

Detailed Description Paragraph Table (5):
TABLE 5 ______________________________________ Data packet header layout Name Type Description ______________________________________ iMagicCookie Int32 Magic number (0.times.BDE01996). Identifies this as a data packet in WINTBL format (little endian). iMajorVer Int16 Major version number (1). iMinorVer Int16 Minor version number (0). iHeaderSize Int32 Size of fixed part of header in bytes (24). iColCount Int16 Column count. Int16 Reserved (0). iRowCount Int32 Row count. (0 if unknown, or if data packet only contains metadata). iProperties Int32 pcklpropsMETADATA.sub.-- INCL (1): Metadata (field descriptors, optional parameters) are included. Column descriptor section.

____

Hit List

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Search Results - Record(s) 1 through 1 of 1 returned.

☐ 1. Document ID: US 5970490 A

```
L7: Entry 1 of 1                File: USPT                   Oct 19, 1999

US-PAT-NO: 5970490
DOCUMENT-IDENTIFIER: US 5970490 A

TITLE: Integration platform for heterogeneous databases
```

Full | Title | Citation | Front | Review | Classification | Date | Reference | | | Claims | KWIC | Draw De

Clear | Generate Collection | Print | Fwd Refs | Bkwd Refs | Generate OACS

Terms	Documents
L6 and L5	1

Display Format: TI Change Format

Previous Page Next Page Go to Doc#

First Hit Fwd Refs
End of Result Set

Generate Collection Print

L7: Entry 1 of 1 File: USPT Oct 19, 1999

DOCUMENT-IDENTIFIER: US 5970490 A
TITLE: Integration platform for heterogeneous databases

Brief Summary Text (8):
Access to HDBs: Language features for multidatabase interoperability include variables which range over both data and metadata, including relation and database names, and expanded view definitions with provisions for updatability. Selected translation or mapping approaches are known and others which are based upon the 5-level architecture are described in Sheth and Larson. Approaches to semantics-based integration and access to heterogeneous DBs where semantic features are used to interrelate disparate data and resolve potentially different meanings are known. Seen from a higher level, what are needed are mediation services among different representations and systems.

Detailed Description Text (3):
The types of heterogeneity which are addressed by the present invention include different schemata, different data models, differences in physical storage and access issues, and differences in semantics and meaning of the data. Such types of structural and semantic heterogeneity have been a primary impediment to effective data interoperability across disparate organizations.

Detailed Description Text (6):
Even when there is no standard model, semantics may be captured in terms of islands of agreement--that is, localized semantic models. Each local model would be self-consistent and utilize a common vocabulary (sometimes this is referred to as an ontology). However, a term in different models or islands may have different meanings. A collection of such islands of agreement may address a substantial portion of the semantics of a large application.

Detailed Description Text (7):
As a simple example, in the United States the use of feet, inches, and miles for length forms one semantic island of agreement, while the use of pounds, ounces, and tons for weight forms another island--these two semantic islands often occur together, but one deals with length while the other deals with weight. Analogous but different semantic islands occur in Europe where meters and centimeters are used for length, and kilograms and grams are used for weight. Even the meaning of the weight `pound` also depends on the semantic island, as there is both the troy system pound (which is 0.373 kg) and the avoirdupois pound (which is 0.454 kg). The word `pound` as a unit of measure also occurs in another semantic island for monetary currency, as in the British pound. Conversion between different semantic islands also can be time dependent, as in the conversion of dollars to yen.

Detailed Description Text (8):
Another example of an island of agreement is the topological map, which takes on a precise meaning in conjunction with a legend that specifies which visual map representations are associated with which topological features. The legend appeals to a commonly agreed upon model of topological maps, and in so doing it makes precise the semantics of the map by specifying the choice of features and visual

representations to be used in that map. In some sense, the legend helps to make a map self-describing.

Detailed Description Text (21):
Once a set of basic self-description attributes, together with an extensible set of additional attributes are agreed upon, the module can list its full set of self-description attributes as one of the pieces of information it provides. So long as the meaning of the extended attributes can be determined by the registry, a substantial amount of operational and semantic information can be conveyed automatically.

Detailed Description Text (24):
Because the self-description information is self-contained within the package itself, it increases the usefulness of the package by not requiring information about it to be scattered around the system in the form of documentation files, source code comments, configuration files, and post-it notes on the screen of the last person that compiled the package. There is a great deal of potential information that can be stored in the self-description, and while there are few servers that utilize that information currently, the system has been designed to provide such information and other metadata for the next generation of intelligent software agents.

Detailed Description Text (40):
The HLDSS specification language is uniform regardless of the diverse database, data structure, and file representations. The significant differences among these different data representations is accounted for by annotations on the HLDSS productions (statements). These annotations give rise to different specialized interpretations of each production, based upon the annotation. The uniformity of the HLDSS language foreshadows the uniformity of the internal intermediate representation of the actual data within the information mediator/bridge 60.

Detailed Description Text (45):
The interpretation of these productions depends upon the optional annotations. If the annotation specifies "RDB:<database.handle>" then the production refers to a relational database table where the left hand side (LHS) of the production, e.g., `Header`, is the table name, and the right hand side (RHS) components are the attributes/columns which are to be selected. Each application of this production to the input would produce one such tuple from the table. The annotation may optionally specify a full SQL query, and this production will then refer to the resulting virtual table.

Detailed Description Text (77):
Referring again the FIG. 2, the HLDSS is parsed and processed by schema analyzers 32 and 52 to create annotated logical structure diagrams (LSDs) 34 and 54, each of which is a schematic structure graph (like a parse tree) that represents the schema and data structures of the data resources. Logical structure diagrams provide a form of meta-schema in that LSD's may be used to graphically describe different data models and schemas. The motivation for the LSD internal representation is that a context-independent uniform graph-based representation can represent all anticipated data resource schemata and structures. The context-dependent interpretation is dictated by annotations, which are processed by the HLDSS preprocessor and the recognizer generator 36 and view generator 56 modules.

Detailed Description Text (78):
Internally, the uniform LSD) structure represents aggregations (e.g. sets, hierarchically nested data, etc.) and associations (n-ary relationships, attributes, dependencies). Basically, the LSD consists of nodes and edges, where each node and edge logically may be associated with both an interpretation and a label, as described below. This interpretation does not change the uniform LSD structure, but rather enables us to see how the LSD corresponds or maps to (or

from) the external heterogeneous formats and structures.

Detailed Description Text (82):
Interpretation of the LSD relative to different data model formalisms, such as a relation, an attribute, an object, or a method which returns a value, etc. Such interpretations impact the meaning of the nodes and edges of the LSD.

Detailed Description Text (84):
These layers of syntax, interpretation, and labeling, enables the use of the same LSD syntax to represent very diverse data models and application schema.

Detailed Description Text (85):
The ability to define language constructs that can be applied to many types of data models and schemas, hinges on the ability to define different interpretations of such uniform constructs. Then the differences among schemas and data models can be encapsulated into such interpretations. Since a given information bridge may access data from multiple sources and/or produce data with multiple destinations, also attached to the LSD for convenience of processing, are the media-specific annotations that are part of the initial HLDSS specification.

Detailed Description Text (87):
1) The descriptive representation language or formalism is the logical structure diagram. It is a logical graph (often a tree) of data entities and logical dependencies. The fact that this descriptive language is uniform in its logical structure depends on the ability to independently specify the interpretation and naming relative to different data models and different databases.

Detailed Description Text (88):
2) The choice of data model, as described in annotations, specifies interpretations such as: whether an association represent a relational table, or a relationship between two objects, or a relationship between an aggregate object and the members of the aggregate. These important differences are accounted for by the interpretation of the LSD constructs.

Detailed Description Text (90):
More precisely, the logical structure diagram (LSD) is defined as a triple (N, E, I), where N is a set of nodes, E is a set of edges over such nodes, and I is the interpretation of the nodes and edges relative to the data model and schema. Each edge E.sub.i is an tuple consisting of k.sub.Ei nodes from N and an optional edge label. The edge represents an association or relationship, and in general may be a hyperedge (thus making the LSD a hypergraph). Binary edges, that is, with two nodes plus an optional edge name also will be used.

Detailed Description Text (91):
The interpretation I=(I.sub.N, I.sub.E) consists of two parts. I.sub.N is an interpretation over the nodes which partitions the node set N into subsets, each corresponding to a primary component type of the chosen data model (e.g., relation table names and attribute names). I.sub.E is an interpretation which partitions the edges so that a subset of edges contains a common set of node(s) from one partition of I.sub.N and this subset of edges relates these node(s) with nodes from a different partition of I.sub.N. Thus the interpretation I represents and makes explicit the primary implicit relationships of the data model.

Detailed Description Text (92):
Thus for example, a relational data model would have an interpretation in an LSD where I consists of nodes which denote relation names, nodes which designate attribute names within the relations, and edges would connect relation nodes and attribute nodes. Similarly, an object model would designate nodes as object types, attributes, or methods, and edges would designate either: 1) a named relationship between an object type and one of its attributes, 2) a subtype (specialization)

relationship between an object type and its parent object type, or 3) an aggregation relationship between an object type and its component object types.

Detailed Description Text (101):
Thus the logical structure diagram (LSD) is a uniform representation of the HLDSS specification and represents the schematic structure of the data. With the source and target annotations, the interpretation of the LSD describes the heterogeneous data resources. When looked at without the annotations, the LSD provides a uniform tree structured description of the intermediate data within the transformer of the Information Mediator. A directed cycle would arise when a data structure refers to itself recursively in its schema or definition. Multiple parents would arise when both parent nodes refer to a single physical shared substructure.

Detailed Description Text (115):
2. Interpretations of database patterns relative to different data model formalisms: object-oriented, relational, etc. This impacts the meaning or interpretation of nodes and edges.

Detailed Description Text (131):
Thus both the extended relational operators and the tree manipulation operators are applicable. Since a transformation is applied to each subtree instance, the alternate interpretations as either a set of tuples or a set of subtrees are dependent upon the application and the type of transformation rule.

Detailed Description Text (146):
The operator can be viewed in either of three related ways. It may be viewed as the traditional relational operator "project", in that each instance to which it is applied can be viewed as a tuple (possibly complex) of the relation named by the second argument; it thus provides relational projection onto the designated columns of the tuple. Alternatively, it may be viewed as SelectByPosition, since it selects the components of the LabeledSubTree according to the indices given in the first argument. Or it may be viewed as a permutation, since it rearranges each first level sequence or subtree of the second argument--it is iteratively applied to each such subtree instance at that level. These interpretations are equivalent operationally, but the interpretation is relative to how the user or integration administrator is viewing the data. Thus there are synonyms for this operator as Project or Permute.

Detailed Description Text (158):
These two functions extend the prior join operations with the three outer join options, of LEFT, RIGHT, FULL, meaning retain unmatched left tuples, or retain unmatched right side tuples, or retain all unmatched tuples from either side. All the tuples actually are represented as LSD instance trees, since they may include complex substructure for the attribute components.

Detailed Description Text (180):
The second form of the cross product rule simply omits the UpTo clause, which changes the meaning of the rule just to the extent that no implicit Collect is performed for the left hand side tokens--the * serves to designate the rule as a cross product. In this case, each of the left hand side token arguments is then taken to already represent a set, as may occur if a left side argument was a Node+ from the input tree or if the data has already been processed by an explicit Collect rule. The cross product then proceeds as before, applying the TransformFunction to each combination of values from the input collections--duplicates are not eliminated.

Detailed Description Text (222):
There are two viable alternative solutions when using a language that does not provide such a facility. The first is to create an internal language and interpreter for the intermediate representation and use that for the description

and interpretation. The drawback of this approach is that the internal representation is limited by the language that is created to describe it. As system requirements change or expand, this representation, as well as the interpreter, must be changed to allow new features to be added. The second option is to generate code in an existing language, compile the code, and then execute it. The draw back is that it requires a separate compilation and execution phases, which represent separate processes. However, the benefit is that since the generated code is the same language as the generator (C++ in this case), it can be directly used and has all the expressive power of the underlying language (C++).

Detailed Description Text (243):
Thus the execution of the parser controller nodes provides the postorder traversal of the tree down to the level of uniform regions. Then the associated parser for the particular type of uniform region is invoked. It traverses the nodes of the uniform region, and constructs a corresponding instance subtree/subgraph. The uniform region parsers typically are specialized, and interpret the LSD schema tree relative to the specific annotations of that region. The nodes and edges may be treated differently in different regions, and even different edges and nodes within a region may have different meanings based upon the annotations. In contrast, each parser controller node treats its immediate descendants uniformly, and this applies recursively downward until a uniform region or terminal node is encountered.

Detailed Description Text (301):
A semantic data specification language (SEMDAL) incorporates features for management of metadata for databases, structured data, document structure, and collections of multimedia documents as well as web-based information. Metadata is descriptive information about data instances. This metadata may provide the schema structure in which the instance data is stored and/or presented. The metadata also may provide a wide variety of other information to help interpret, understand, process, and/or utilize the data instances. At times, metadata may be treated as if it were instance data, depending upon the application.

Detailed Description Text (303):
While a subset of SEMDAL translates into the HLDSS, a somewhat different but significantly overlapping subset translates into SGML and thus offers a different surface syntax. Thus, it is the interpretation associated with the syntax that is essential. Normally, SGML is interpreted to refer to document structuring with tags (markup) and HLDSS refers to databases and structured data. However, with SEMDAL different interpretations may be associated to a SEMDAL construct to achieve either of these results. This facilitates bridging the gap between structured databases and free form text and documents.

Detailed Description Text (304):
It is important to be able to make the interpretation explicit and to be able to apply alternative interpretations within a specification language. The interpretation is defined as the operational processing which is to be applied to different constructs of the syntactic language, both in terms of the logical meaning as well as the procedures which are to be applied for a particular application. The following is an example of a Frame construct:

Detailed Description Text (308):
Note that the ATTributes are scoped within that Frame, that attributes defined in the !ATT portion are semantic attributes, and that the names on the right side of the ": :" are structural components or structural attributes. Any representation language includes in its definition some implicit interpretations. What is important is that the set of interpretations which are being described are explicit so that different interpretations can be captured within the same language by specifying the relevant interpretations rather than needing different languages for each.

Detailed Description Text (309):
In other words, if an ATTribute is added in the above Frame with the INTERPretation that this is a `document` then the right hand side (after ": :") will be interpreted as components of the document, consisting of a header, a bound (read as an `abstract`), textural nodes (e.g., paragraphs), and a count of the paragraphs. When this interpretation is specified, the !Frame construct is very similar to the !Element construct of SGML.

Detailed Description Text (311):
If, however, INTERPretation is specified as a `relational database`, then in the previous example Frame, manifest is a relation which has fields/attributes for header, bound, nodes and numNodes. In fact, what is referred to as the `phyla` in the HLDSS for database interoperability is essentially what is meant here by the interpretation. This is what determines whether the HLDSS statement is to be treated as a reference to a relation, an object, or part of a parsing specification for a structured file.

Detailed Description Text (312):
Of course certain semantic ATTributes, such as type, will have meaning under just one INTERPretation or the other. Thus the INTERPretation attribute can be used by a reasoning or processing engine to determine how the other attributes and constructs of the specification should be interpreted and utilized.

Detailed Description Text (313):
The notion of interpretation enables the use of the same representation to express a matrix. This is one of the novel aspects of how interpretation is used in the SEMDAL language to unify and express seemingly diverse constructs such as structured documents, relational and object-oriented databases, and matrices.

Detailed Description Text (314):
In the following example, a 2-dimensional matrix is defined with a value that is called a WindVector. The structural components of WindVector are defined in a second Frame. "MatA" is a matrix because its INTERPretation is "matrix". This means that the right most structural component is the value at each entry of the matrix, and the other structural components are the dimensions, thus it is a 2-dimensional matrix. Hence the dimensions and value attributes could have been deduced.

Detailed Description Text (318):
N-ary relationships are also achieved as another interpretation of a Frame. Note that the !ATTribute specifications within a Frame define only ternary relationships between the component, the attribute name and the value for that attribute. For example in <!ATT Time Units=seconds>the component is Time, the attribute is Units, and the value is seconds.

Detailed Description Text (319):
An interpretation of a Frame as a `relation` provides the left hand component (before the ": :") as the name of the relation and the right hand components as the components that participate in the relation. Normally a single construct in other specification languages could not represent both the substructure of a document and an n-ary relationship with the same kind of specification statement, but here this is accomplished easily through the novel use of an INTERPretation.

Detailed Description Text (321):
The semantic attribute INTERPretation as a `relation` means that this Frame represents the Supplies relationship between Supplier, PartNo, ShopLocation and Cost. The Semantic attributes also indicate that Cost is given as whole dollars (integer) in either US currency or Canadian currency. In the !ATT clause either "=" is used to set a default value or an "IN" comparator to constrain the value to an enumerated set of possible values or a range. An attribute also can be defined without specifying its default value. Other comparators would be expressed as

constraints. If it is desired to specify a constraint by "IN" as well as to set a default value, two expressions would be used, for example:

Detailed Description Text (323):
Note that the above INTERPretation as a `relation` is not the same as a relational database, since here it has not been committed how this 4-ary Supplies relation is to be materialized and stored. This relationship could be, if one wanted, represented by Horn logic clauses in a prolog system.

Detailed Description Text (325):
The meaning of the variable is designated by setting it equal to the components involved. Use of the wildcard "*" can refer to multiple components, which may be further restricted if desired by a predicate that the variable also must satisfy. Thus valid expressions for defining variable include:

Detailed Description Text (334):
Constraints are a very important part of metadata for the following reasons:

Detailed Description Text (339):
5. Constraints are used to make the active metadata repository respond to changes and events and to initiate actions dynamically.

Detailed Description Text (355):
One of the significant advances made by the present invention is the seamless ability to store diverse forms of metadata in a single repository utilizing a single logical and physical storage scheme. The differences between metadata representations as currently used are accounted for via the operational interpretation of the uniformly stored information. Furthermore, alternative syntactic realizations of this metadata are provided so as to continue to support those applications which expect different syntax for different uses.

Detailed Description Text (356):
The logical design level of the metadata repository allows for multiple implementations and is convenient in different storage architectures. The physical storage of one implementation is relation-based in order to be accessible with the Java Database Connectivity (JDBC) standard for the Java interfaces to relational databases. Note that the choice of repository implementation is independent from the data models being described.

Detailed Description Text (357):
The metadata repository of the present invention encompasses both structure and semantics, and it does so in a way that can accommodate separately developed and different metadata within a primary framework.

Detailed Description Text (367):
The metadata representation of the present invention shall be referred to as the Semantic Metadata Description Language, or SEMDAL and is also referred to as MDS (MetaData Specification) for short. It is a frame-based representation where a MetaFrame contains potentially multiple attribute value specifications. Each attribute may have subattributes, and these too may have subattributes if needed. This creates a potentially hierarchical array of descriptors. A simple example of such descriptors are `units` and `precision` information for length, weight, or other measures.

Detailed Description Text (368):
The potential hierarchy of semantic descriptors, and the actual substructure of the stored data are distinguished as described by the MetaFrame's metadata. Such data substructure is explicit as in SGML, and is denoted by a `Substructure` attribute (similar to SGML's `!Element`), or by using a syntactic shorthand similar to BNF (Backus-Naur Form) grammar specifications.

Detailed Description Text (379):
The underlying logical model for the metadata repository thus is a hierarchical structural model that can represent the spanning tree over a graph or directed graph structure. The logical model for semantic attributes provides a hierarchy of subordinate descriptors--this semantic hierarchy is separate from the structural hierarchy. Each semantic descriptor can be represented as a whole, with operations to retrieve the associated value, the parent descriptor which is refined by this subattribute, and the set of subattributes of the current descriptor.

Detailed Description Text (384):
The representation for an analog watch then can refer to Clock-Schema.SecondHand.Length.Units to define the units in which the length of the second hand will be expressed. Then a simple number in the actual data representation will have meaning that is explicit. Such units is semantic information and is part of the metadata that is managed, and it is necessary information in order to properly utilize the actual length data values--which may be stored in a separate database of instance data.

Detailed Description Text (397):
Each of these are referred to as different interpretations of the same syntactic representation. SEMDAL provides a system level attribute `INTERP` to capture this INTERPretation, indicating, for example, whether the data is from a relational database table or from the data members of an object.

Detailed Description Text (398):
Note that hierarchical structure of the data is defined using "Substructure" attributes or its abbreviation as"::". In contrast, subattributes, such as "units", do not define substructure of the data--rather these subattributes provide additional metadata to help interpret the data which is presented.

Detailed Description Text (424):
Attributes such as INTERPretation are system defined attributes. In order to avoid name conflicts as the application designer creates attribute names, system attributes may be prefixed with the "!" exclamation mark, as "! INTERP".

Detailed Description Text (425):
The logical structure described for the Metadata repository is hierarchical, which could be naturally implemented in a `nested relational` database or an object database, as well as in other databases. Due to the increasing value of the JDBC connectivity between Java programs and relational databases, a relational implementation has been chosen. Thus the Metadata Repository will be accessible via the JDBC standard API. The metadata which are stored describe related instance data--which usually is stored separately in one or more repositories, such as in a relational or object database, document repository , or multimedia digital library.

Detailed Description Text (441):
A structure path expression may begin with a FrameName and consists of dot-separated terms, each term being a LHS or RHS, such that the RHS must correspond to that LHS. If this LHS also appears as a RHS, the structure path may continue with a corresponding LHS. Thus in the MDS1 example on page 8, the structure path "Manifest.bound.scaleX" could be used to refer to this structure, the associated value, as well as any other semantic attributes of scaleX. Also, a structure path expression ending in a non-terminal can be used to refer to the data structure instance(s) corresponding to this non-terminal--each instance itself being a data structure consisting of the multiple data values and their associated structure, as described by the MetaFrame productions. Structure path expressions can be used in a generic uniform query language to refer to data in any form of structure or database, based upon the metadata in the repository.

Detailed Description Text (454):
Sometimes the descriptive metadata is of the same nature for each entry. For example, the metadata for a relational database usually will consist of the same kinds of descriptive attributes for each relational table and for each column, just with different values. In these cases a more compact representation as a table of Metadata values is possible--though doing imparts no additional information. That is, if the kinds of semantic attributes are the same for each, the same attribute subpath does not need to be repeated for each different table and value. Rather, each attribute subpath could be taken as a column name in a meta-table to provide greater conciseness.

Detailed Description Text (455):
It should be noted that the metadata framework can be used to describe the structure and semantics of the MetaFrame representation as well as the metadata implementation vehicles of MetaFrameAttributes and MetaFrameStructure tables. Thus the metadata repository is self-describing. This observation helps to address notions of "meta-meta-data". Specifically, since the metadata framework developed is self-describing, there is no need for further levels of "meta-meta" representation formalisms.

Detailed Description Paragraph Table (6):
<MetaFrame MDS1 ; Manifest :: header+ bound+ Photo ; header :: Name Version Type Fmt ; bound :: numElements numNodes scaleX scaleY ; bound.INTERP = "ODB" ; bound.System = "Ontos" ; bound.OSQL = "Select Mesh.bound ..." ; header . INTERP = "RDB" ; .System = "Sybase" ; .Site = "@db.foo.com:42" ; .Access = "Fred" ; (Name.vertline.Version).(Type="string", Length=40) ; Version.Default ="1.0" ; Photo.(Type="Image", Encode="JPEG", Size=480, Location="NYC", Title="Metadata Architecture...", Date="6/5/97", SequenceNo=34, Size.Units ="kB") ; /MetaFrame>

CLAIMS:

2. The method as claimed in claim 1, wherein the first high level data structure specification and second high level data structure specification comprise:

diverse metatadata, including semantic metadata.

5. The process as claimed in claim 1, wherein parsing further comprises:

forming nodes and edges of the first logical structure diagram, each node and edge being logicially associated with a label and an interpretation, the label of a specific logical structure diagram component corresponding to a particular application schema component and the interpretation of the logical structure diagram impacting the meaning of the nodes and edges of the logical structure diagram and being derived from the annotations in the high level data structure specification to enable using the same high level data structure specification syntax and the same logical structure diagram contsructs to represent diverse data models and application schema.

9. A method for integrating heterogeneous data embodied in computer readable media having source data and target data comprising:

providing an interoperability assistant module with specifications for transforming the source data;

transforming the source data into a common intermediate representation of the data using the specifications;

transforming the intermediate representation of the data into a specialized target representation using the specifications;

creating an information bridge with the interoperability assistant module through a process of program generation;

processing the source data through the information bridge; outputting the target data, wherein the target data is in a non-relational form with respect to the source data; and

outputting the target data, wherein the target data is in a non-relational form with respect to the source data,

wherein providing the interoperability assistant with specifications comprises:

inputting a first high level data structure specification which describes the source data representation;

inputting a second high level data structure specification which describes the target data;

inputting a high level transformation rule specification;

processing the first high level data structure with a first schema analyzer and a recognizer generator to generate a source recognizer module;

processing the second high level data structure with a second schema analyzer and a view generator to generate a target builder module;

processing the high level transformation rule specification with a transformer generator to generate a transformer module;

parsing the first high level data structure specification with the first schema analyzer to create an annotated logical structure diagram, the logical structure diagram serving as a schematic structure graph that represents the logical relationships of the source data in a context-independent uniform manner; and

parsing the second high level data structure specification with the second schema analyzer to create an annotated logical structure diagram, the first logical structure diagram serving as a schematic structure graph that represents the logical relationships of the source data in a context-independent uniform manner,

wherein parsing the first and second high level data structure specifications includes forming nodes and edges of the first logical structure diagram, each node and edge being logically associated with a label and an interpretation, the label of a specific logical structure diagram component corresponding to a particular application schema component and the interpretation of the logical structure diagram impacting the meaning of the nodes and edges of the logical structure diagram and being derived from the annotations in the high level data structure specification to enable using the same high level data structure specification syntax and the same logical structure diagram constructs to represent diverse data models and application schema.

10. The method as claimed in claim 9, wherein the first high level data structure specification and second high level data structure specification comprise: diverse metadata, including semantic metadata.

15. The method as claimed in claim 14, wherein the first high level data structure specification and second high level data structure specification comprise:

diverse metadata, including semantic metadata.